1. 上课须知

课程主题：史诗级宇宙最全30条Hive性能调优全详解
上课时间：20:00 - 23:00
课件休息：21:30 左右 休息10分钟
课前签到：如果能听见音乐，能看到画面，请在直播间扣 666 签到

2. 上次内容总结

Hive 的企业应用技巧总结分析和实战

```
1、Hive的架构设计和SQL语句复习总结
2、Hive的窗口分析函数详解
3、Hive的JSON，Transform语法，复杂数据类型处理解读
4、Hive的三大典型需求分析和企业案例完整实现
    自连接（自连接 窗口函数）
    TopN（explode，  leteral view，窗口函数）
    行列转换（case ... when ... else... end）
5、Hive的典型常见面试题型分析和思路总结
    8个点
6、Hive全局排序（彩蛋）方案的最大问题：数据倾斜
    范围分区（数据分段存储：第一段 > 第二段 > 第三段 .... ） + 局部排序
    sort by + order by
```

3. 上次作业

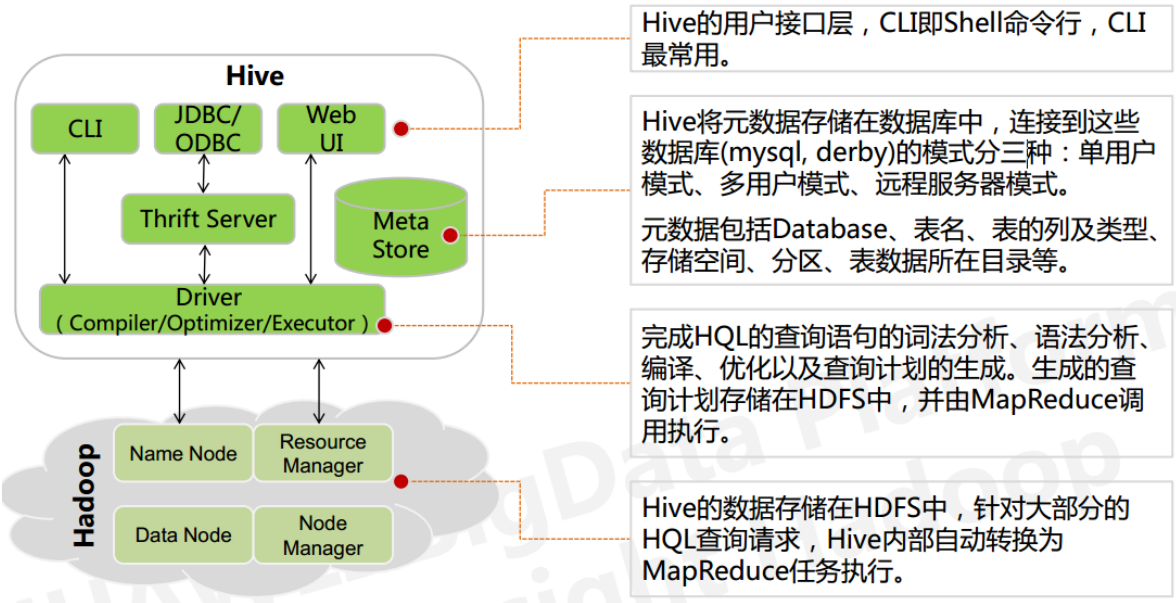
见文档：Hive最终大作业.pdf

4. 本次内容预告

```
1、深度剖析Hive架构设计和工作原理
2、史诗级Hive性能调优30条最佳实践
3、超经典Hive调优企业级案例实践
```

4.1. 深度剖析Hive架构设计和工作原理

4.1.1. Hive 架构设计



Hive内部四大组件：**Driver驱动器：编译器Compiler，优化器Optimizer，执行器Executor**

4.1.2. Hive 工作原理

Driver组件完成HQL查询语句从词法分析，语法分析，编译，优化，以及生成逻辑执行计划的生成。生成的逻辑执行计划存储在HDFS中，并随后由MapReduce调用执行。

最重要的结论：Hive是基于Hadoop的数仓工具，把存储在 HDFS 上的结构化数据看做是一张二维表格，提供类似于 SQL 的操作方式，简化分布式应用程序的编写，底层实际上还是运行 MapReduce 程序，所以你在对待 Hive 的 SQL 的优化的时候，一定要注意，你其实在优化 MapReduce，而不是把 MySQL 中的 SQL 优化技巧生搬硬套上来。

4.2. 史诗级Hive性能调优30条最佳实践

4.2.1. 调优概述

Hive 作为大数据领域常用的数据仓库组件，在平时设计和查询时要特别注意效率。**影响 Hive 效率的几乎从不是数据量过大，而是数据倾斜、数据冗余、Job或I/O过多、MapReduce 分配不合理等等。**对 Hive 的调优既包含 Hive 的建表设计方面，对 HiveHQL 语句本身的优化，也包含 Hive 配置参数 和 底层引擎 MapReduce 方面的调整。

```
1、Hive 的建表设计
2、HiveHQL 语句本身的优化
3、Hive 配置参数
4、底层引擎 MapReduce 方面的调整
```

所以此次调优主要分为以下四个方面展开：

```
1、Hive 建表设计层面
2、HQL 语法和运行参数层面
3、Hive 架构层面
4、Hive 数据倾斜
```

总之，Hive调优的作用：在保证业务结果不变的前提下，降低资源的使用量，减少任务的执行时间。

多讲一句话：

```
1、Hive 的底层 SQL 其实运行的是 mapreduce，不适用于 MySQL 的调优
2、所谓的调优：在资源固定不变的情况下，不够用了，才需要调优！一切的调优都比不上直接增加资源
```

4.2.2. 调优须知

1、对于大数据计算引擎来说：**数据量大不是问题，数据倾斜是个问题。**

2、Hive的复杂HQL底层会转换成多个MapReduce Job并行或者串行执行，**Job数比较多的作业运行效率相对比较低**，比如即使只有几百行数据的表，如果多次关联多次汇总，产生十几个Job，耗时很长。原因是 MapReduce 作业初始化的时间是比较长的。

3、在进行 Hive 大数据分析时，常见的聚合操作比如 sum，count，max，min，UDAF 等 ，不怕数据倾斜问题，MapReduce 在 Mapper 阶段 的**预聚合**操作，使数据倾斜不成问题。

4、好的**建表设计，模型设计**事半功倍。

5、设置**合理的 MapReduce 的 Task 并行度**，能有效提升性能。比如，10w+数据量 级别的计算，用 100 个 reduceTask，那是相当的浪费，1个足够，但是如果是亿级别的数据量，那么1个 Task 又显得捉襟见肘

6、**了解数据分布**，自己动手解决数据倾斜问题是个不错的选择。这是通用的算法优化，但算法优化有时不能适应特定业务背景，开发人员了解业务，了解数据，可以通过业务逻辑精确有效的解决数据倾斜问题。

7、数据量较大的情况下，慎用 count(distinct)，group by 容易产生倾斜问题。

8、对**小文件进行合并**，是行之有效的提高调度效率的方法，假如所有的作业设置合理的文件数，对任务的整体调度效率也会产生积极的正向影响

9、优化时把握整体，**单个作业最优不如整体最优。**

10、优化 MySQL 的 SQL优化技巧，是不适用在 Hive 的。

4.2.3. 具体调优

见文档：Hive调优文档.pdf

4.3. 超经典Hive调优企业级案例实践

4.3.1. 第一个例子：日志表和用户表做链接

背景：

```
1、小表 join 小表： 无论怎么做，都很容易实现
2、大表 join 小表： mapjoin 使用内存资源帮助免去了reducer阶段，提高效率
    如果这个小表不满足条件，不是特别的小：不大不小
    1、拆分小表，成多个更小的小表，做多次mapjoin
    2、因为一般来说，大表joini小表，大表就是事实表，小表就是维度表。
    通用的规律：并不是所有的维度ID，都在事实表中有存在！
    例子：支付宝：10E： 每个用户每天都有交易么？
3、大表 join 大表： 最难的方案
```

简单来说：就是事实表（大表） 和 维度表（小表）做链接。怎么优化?

```
select * from log a left outer join users b on a.user_id = b.user_id;
```

分析数据：

```
1、users表中，并不是所有的用户，都会产生日志。例如：百度的所有用户并不是所有都有搜索记录
2、如果某个用户使用了百度，大概率今天的搜索记录不是一个两个，而是很多
```

关于 Hive 中的 Join，最常见的两种 Join 实现方案，就是 ReduceJoin 和 MapJoin，使用 MapJoin 来解决上面的问题！发现一个问题：这个小表的数据量并不是特别的小，显得不是特别合适。如果使用 ReduceJoin，则有可能出现数据倾斜。

需求场景：

```
users 表有 600w+ （假设有5G）的记录，把 users 分发到所有的 map 上也是个不小的开销，而且 MapJoin 不支持这么大的小表。如果用普通的 join，又会碰到数据倾斜的问题。
```

合适的解决方案：

```
1、给 users 表做过滤，但是过滤需要条件， 先得到过滤的 userid
select distinct user_id from log;  ===> result_a

2、再获取有日志记录的用户的信息
select /*+mapjoin(a)*/ b.* from result_a a join users b on a.user_id =
b.user_id;  ===> result_c

3、让 c 表和 log 表做 mapjoin
select /*+mapjoin(c)*/ c.*, d.* from result_c c join log d on c.user_id =
d.user_id;
```

改进方案：

```
select /*+mapjoin(x)*/ * from log a
left outer join (
    select /*+mapjoin(c)*/ d.*
       from ( select distinct user_id from log ) c join users d on c.user_id =
d.user_id
) x
on a.user_id = x.user_id;
```

假如，log 里 user_id 有上百万个，这就又回到原来 MapJoin 问题。所幸，每日的会员 uv 不会太多，有交易的会员不会太多，有点击的会员不会太多，有佣金的会员不会太多等等。所以这个方法能解决很多场景下的数据倾斜问题。

谓词下推 + where ： 不参与计算的时间，能早点过滤就尽量早点过滤掉！

4.3.2. 第二个例子：位图法求连续七天发朋友圈的用户

```
1、内存 + 磁盘
2、范围分区
3、位图 + 布隆过滤器
4、索引
5、LSM-Tree
```

每天所有的用户发的发朋友圈（1、一条朋友圈一条记录。 2、一天的所有朋友圈建一张表来保存）

```
1、实现形式：log a join log b join log c ..... join log g
2、链接条件：userid
3、这个需求实现的难度：7张大表做链接
```

原有的模式是：

```
1、每个用户发了一条朋友圈，就会记录这条朋友圈的信息
2、优化数据存储（使用位数组来表示）：当天发了朋友圈的所有用户，使用1来表示，没有发朋友圈的使用0来表示，整个微信的所有用户是否发朋友圈的记录就是一串非常长的 01010 字符串
```

```
在某一天里面，某个用户是否发了朋友圈，使用0和1来表示：
1、第一个文件：代表第一天
    10101010110101010101011010
2、第二个文件：代表第二天
    10110101101010101001011010101
3、第三个文件：代表第三天
    10101010110101010010110101010
4、第四个文件：代表第四天
    10101010110101010010110101010
....
最后七个文件按位求与！
```

每天都要求 微信朋友圈 过去连续7天都发了朋友圈的小伙伴有哪些?

假设每个用户每发一次朋友圈都记录了一条日志。每一条朋友圈包含的内容：

```
日期，用户ID，朋友圈内容.....
dt, userid, content, .....
```

如果 微信朋友圈的 日志数据，按照日期做了分区。

```
2020-07-06  file1.log(可能会非常大)
2020-07-05  file2.log
.......
```

实现的SQL：

```
// 昨天和今天 = 结果1
select a.userid from table a join table b on a.userid = b.userid;

// 结果1 join 前天 = 结果2
// 上一次join的结果 和 前天 join
.....

// 上一次join的结构 和 大前天 join
.....
```

好的解决方案：位图法 BitMap

```
假设微信有10E用户，我们每天生成一个长度为10E的二进制数组，每个位置要么是0，要么是1，如果为1，代表该用户当天发了朋友圈。如果为0，代表没有发朋友圈。

每天的数据总量：10E / 8 / 1024 / 1024 = 119M左右

求Join实现：两个数组做 求且、求或、异或、求反、求新增
1、求过去7天发朋友圈的用户有哪些？          按位求或
2、求过去7天都发了朋友圈的用户有哪些？      按位求与
```

5. 最强大脑

挑战：现在有50对情侣每隔3秒从我面前走过，他们走过的时候，都会告诉我，他们结婚了没有！

10101011010101011010101011001011010101010

a31d68234863

问题；第34对结婚了么？（编解码的速度足够快）

6. 总结

大致一些总结如下：

```
1、资源不够时才需要调优
    资源足够的时候，只需要调大一些资源用量
2、业务优先，运行效率靠后
    首先实现业务，有多余精力再考虑调优
3、单个作业最优不如整体最优
    全局最优
4、调优不能影响业务运行结果
    业务正确性最重要（combiner: avg ）
5、调优关注点
    架构方面
    业务方面
    开发方面
    资源方面
```

7. 作业

见文档：Hive最终大作业.pdf